MINIMALIST Highlights

July 19, 1999

This document provides a quick run-through of some of MINIMALIST's capabilities. It is not intended as a reference manual. For more information, see the online help (type "`help`" at the MINIMALIST command prompt). More complete documentation will be provided at a later date.

1 Installation

First, choose a directory into which you'll install the software. This will require roughly 20 Mbytes of disk space. The installation includes examples and synthesis scripts (as well as the core software itself). In the examples that follow, you will be running MINIMALIST from within one of the example directories.

In the following, the installation directory will be referred to as *InstallDir*. Commands you should type as-is are given ***in bold type, like this***.

1.1 Retrieving the Distribution using FTP

As of this writing (July 1999), binaries are available for Solaris 2.x; distributions for Linux and AIX will be made available shortly. If you need a distribution for another platform (e.g. HP/UX), let us know.

```
$ cd InstallDir
$ ftp ftp.cs.columbia.edu
Name (ftp.cs.columbia.edu:user): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: user@host
ftp> cd pub/rmf
ftp> dir
-rw-r--r--   1 2273     async      4878453 Mar 13 01:27 MINIMALIST-Solaris2.tgz
ftp> binary
ftp> get MINIMALIST-Solaris2.tgz
...
ftp> quit
$
```

Substitute the file name retrieved above for `InstallArchive` in what follows.

1.2 Unpacking the Distribution

Simply move to the directory into which you want to install MINIMALIST, and then uncompress and extract the tar archive, like so:

```
$ cd /usr/local/minimalist
$ gunzip < InstallArchive | tar -xvf -
```

1.3 Setting up the Environment

You must do one of the following each time you start up a shell used to run MINIMALIST (or add the appropriate entry to your .profile, .login, or .bashrc):

```
$ source InstallDir/bin/setup.bash                     # Setup bash shells
$ . InstallDir/bin/setup.ksh                            # Setup ksh shells
$ source InstallDir/bin/setup.csh                   # Setup csh/tcsh shells
```

N.B.: running the script directly (i.e. by omitting the "`source`" or "`.`") will *not* have the proper effect. Also, you do *not* need to run this script each time you run `MinShell`.

2 Run-through, Part I - Framework

The first part of the run-through demonstrates the MINIMALIST framework by issuing individual commands to the MINIMALIST's command line interpreter.

2.1 Complete synthesis session

```
$ MinShell                                       # Start the MINIMALIST shell
minimalist> help                                     # Show list of commands
    assign-states           break            call              cd
         continue          define            echo            expr
              for            help              if    impymin-logic
    make-testable       min-logic      min-states             pwd
             quit       read-spec             set    set-encoding
  set-state-cover   show-encoding      show-logic          source
     verify-logic           while      write-flow   write-instant
       write-spec  write-symbolic     write-trans
minimalist> pwd                                     # Show current directory
/u/minimalist/demo
minimalist> ls                                                    # Run 'ls'
bin  ex   lib
minimalist> ls ex
dme-e       dram-ctrl    pe-send-ifc  scripts     stetson
dme-fast-e  hp-ir        pscsi        scsi-iccd92
minimalist> cd ex/dme-e                           # Move to another directory
minimalist> ls
dme-e.bms
minimalist> help read-spec                        # Show syntax of 'read-spec'
        read-spec [-v] <file> [<spec-var>]

Read the Burst-Mode specification in <file> and store it in <spec-var>,
or, if <spec-var> is not specified, 'theSpec'.
minimalist> read-spec dme-e.bms                # Read the Burst-Mode specification
Specification passed validity check.
```

```
Specification has 3 inputs, 3 outputs, and 8 states.
minimalist> plot_graph dme-e.bms
[... a window appears which displays the burst-mode graph ...]
[...     press Ctrl-Q or the Quit button to dismiss it    ...]
minimalist> min-states                                   # Perform state minimization
*** Performing state minimization... ***
There are 5 compatibles.
Forming covering matrix...
Solving state covering problem...
State covering problem solved.
Refining state cover to a partition...
State cover:
 S0 S1 S4
 S2 S3
 S5 S6 S7
Machine has 3 states after minimization.
minimalist> assign-states -F                   # Assign states with a CRF encoding
No encoding style specified; defaulting to critical race-free
*** Performing state assignment... ***
Invoking 'chasm' as 'chasm -C DME_E-F.func DME_E-F.trans'

*** Machine encoded by CHASM ***

State S0': 11
State S1': 10
State S2': 00
minimalist> min-logic -F -L                    # Produce the logic implementation
*** Performing logic minimization... ***
Invoking 'hfmin' as 'hfmin -P -C -l -S -o DME_E-FL.sol DME_E-FL.pla DME_E-FL.btrans'

*** Final PLA ***

# PLA file for machine DME_E-FL
.i 8
.o 5
.ilb LIN RIN UIN LOUT_i ROUT_i UOUT_i Y0 Y1
.ob y0 y1 LOUT ROUT UOUT
.type fr
.p 7
10-----1 00010
-01----1 00010
-0-----1 01000
10-----0 10100
00----1- 11000
--0---1- 10000
-01----0 00001
.e
Number of products:       7
```

```
Total number of literals: 28
Number of products implementing outputs:      4
Literals in products implementing outputs:    17
Literals in products implementing next-state: 16
Average literals per output:                  5.66667
Result stored in 'DME_E-FL.sol'.
minimalist> verify-logic -F -L                         # Verify the implementation
Using logic implementation stored in 'DME_E-FL.sol'.

*** Starting Burst-Mode machine simulation. ***

Implementation verified successfully!
minimalist> plot_graph DME_E-FL.sol                    # Display the implementation
[... a window appears which displays the 2-level logic ...]
[...   press Ctrl-Q or the Quit button to dismiss it   ...]
                                   # Run the above synthesis steps using a script
minimalist> source ../scripts/script-FBO.MOL-CRF dme-e.bms
[...]
minimalist> quit
```

2.2 Session Summary

1) ***MinShell***
2) ***cd ex/dme-e***
3) ***read-spec dme-e.bms***
4) ***plot_graph dme-e.bms***
5) ***min-states -F***
6) ***assign-states -F***
7) ***min-logic -F -L***
8) ***verify-logic -F -L***
9) ***plot_graph DME_E-FL.sol***
10) ***source ../scripts/script.FBO-MOL-CRF dme-e.bms***
11) ***quit***

3 Run-through, Part II - Exploring Synthesis Trade-offs

In this section, we explore some of the trade-offs inherent in the asynchronous synthesis problem, e.g., trading larger area for increased performance.

3.1 AREA: To feed back outputs or not?

```
$ cd InstallDir/ex/scsi-iccd92
$ MinShell
minimalist> source ../scripts/script.noFBO-MOL-EXACT tsend-csm.bms
[... observe total literal count in resulting logic ...]
minimalist> source ../scripts/script.FBO-MOL-EXACT tsend-csm.bms
[... observe better total literal count in resulting logic ...]
minimalist> quit
```

From here on in, we perform similar synthesis runs by invoking `MinShell` with parameterized batch scripts.

3.2 AREA vs. PERFORMANCE: Multi-output, Output-disjoint, and Single-output

$ ***MinShell ../scripts/script.FBO-MOL-EXACT tsend-csm.bms***
`[... observe total literal count and output literal count ...]`
$ ***MinShell ../scripts/script.FBO-ODL-EXACT tsend-csm.bms***
`[... higher total literal count but better average output literals ...]`
$ ***MinShell ../scripts/script.FBO-SOL-EXACT tsend-csm.bms***
`[... higher total literal count, even better avg output literals ...]`

3.3 PERFORMANCE: Effect of Encoding

3.3.1 EXACTLY OPTIMUM OUTPUT LOGIC — Single-Output Product Minimization

$ ***MinShell ../scripts/script.FBO-SOP-CRF isend-bm.bms***
`[... observe output product count ...]`
$ ***MinShell ../scripts/script.FBO-SOP-EXACT isend-bm.bms***
`[... optimum output product count, but longer codes ...]`

3.3.2 OPTIMAL OUTPUT LOGIC — Single-Output Literal Minimization

$ ***MinShell ../scripts/script.FBO-SOL-CRF isend-bm.bms***
`[... observe output literal count ...]`
$ ***MinShell ../scripts/script.FBO-SOL-EXACT isend-bm.bms***
`[... fewer output literals, but longer codes ...]`

3.3.3 TRADING OUTPUT PERFORMANCE FOR CYCLE TIME — Fixed-Length Encodings

$ ***MinShell ../scripts/script.noFBO-SOL-FIXLEN tsend-csm.bms 3***
$ ***MinShell ../scripts/script.noFBO-SOL-FIXLEN tsend-csm.bms 4***
$ ***MinShell ../scripts/script.noFBO-SOL-FIXLEN tsend-csm.bms 5***
$ ***MinShell ../scripts/script.noFBO-SOL-FIXLEN tsend-csm.bms 6***
`[... observe output and next-state literal count getting better ...]`

3.4 HEURISTIC LOGIC MINIMIZATION: Very Large Designs

$ ***cd InstallDir/ex/stetson***
$ ***MinShell ../scripts/script.p2-HFMIN***
`[... observe speed of logic minimization ...]`
`[...  Ctrl-C out of HFMIN and MinShell   ...]`
$ ***MinShell ../scripts/script.p2-Espresso-HF***
`[... much faster logic minimization!!! ...]`